Pooled Tables

A High Performance Batch Reporting Option

A White Paper by Robert Freeman and Sara Elam

Information Builders

Executive Summary

Pooled Tables is an easy-to-use, high-performance feature that reduces your FOCUS reporting costs and frees system resources for both new and existing reporting applications. It delivers immediate and significant benefits to large batch reporting applications that extract data from large databases, with little or no change to the applications.

Once you turn on this option, FOCUS executes multiple report requests in a single pass of the database, reducing total database I/O, CPU time, and elapsed time for the entire pool of requests. You do not have to be familiar with an application or alter its logic to invoke this optimization process. Pooled Tables dynamically analyzes the application and combines reports into a single pool.

Pooled Tables optimizes the data retrieval portion of a report request without affecting output processing. Therefore, pooled requests can produce the full range of FOCUS-supported output formats, including tabular reports, graphs, and extract files.

With Pooled Tables, you can pool requests from any data source accessible to FOCUS. You can access legacy databases such as VSAM, IMS, IDMS, and ADABAS, and relational databases such as DB2, SQL/DS, Oracle, and Teradata, as well as FOCUS databases, sequential files, and joined structures.

Benchmark results document reductions in database I/O, CPU time, and elapsed time for as few as two reports and as many as 25 reports of every size and from every type of data source. See *Benchmarks* for a detailed description of these results. You may be able to pool even more requests than the benchmark tests demonstrated.

Pooled Tables is already installed in FOCUS 7.0.8. You enable the feature by editing the FOCUS initialization file. Changes are effective immediately, without compiling or linking. An IPL is not necessary, and Pooled Tables requires no maintenance or monitoring.

You do not need to understand how Pooled Tables works to make Pooled Tables work for you. Applications that do not take advantage of pooling are not adversely affected with the feature enabled.

To learn more about Pooled Tables, contact your local Information Builders office, visit our World Wide Web site at www.ibi.com, or, in the U.S. and Canada, call (800)969-INFO.

Benefits

Pooled Tables can help you reduce in-house computing costs and external charge-backs, defer the purchase of a larger computer, and minimize reliance on outsourcing.

In any organization, development resources are not always available. Pooled Tables can help reduce costs even if your developers have no time to rewrite applications for performance or have moved on to other responsibilities.

Applications That Benefit From Pooled Tables

Pooled Tables is primarily designed to reduce costs for large batch reporting applications. However, any application that executes two consecutive reports against the same data source is a candidate for its performance benefits.

Data warehousing and Decision Support Systems typically generate several reports using the same data at different levels of aggregation, making them ideal candidates for combined retrieval processing. Cyclical systems (such as month-end and year-end reporting systems) and online applications that generate multiple reports also benefit from Pooled Tables.

Reporting costs are reduced any time that multiple reports can run concurrently. Applications benefit from Pooled Tables when they contain two or more contiguous reports against the same database and have significant I/O costs compared to formatting costs. Applications that access large databases, complex structures, or tape files also benefit from reducing retrieval to a single pass of the data source.

Applications That Do Not Benefit From Pooled Tables

Not all applications are appropriate candidates for Pooled Tables. However, implementing Pooled Tables will not negatively impact any application.

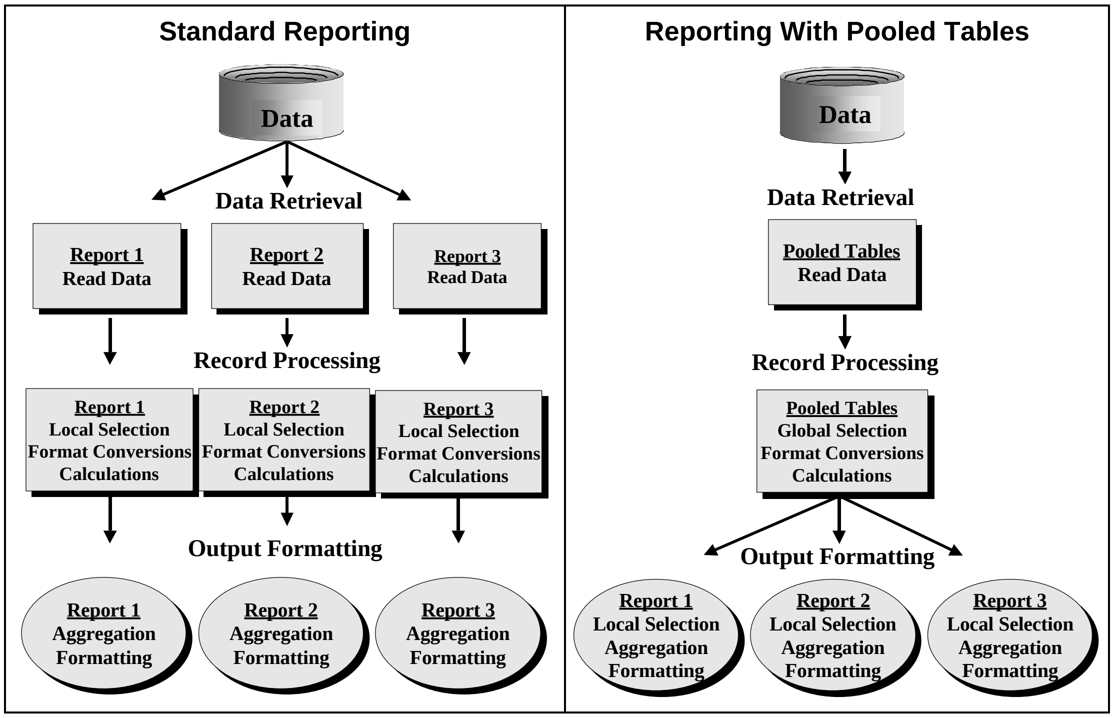
Reports against small databases do not have significant database I/O costs and will see only marginal improvement. Highly tuned applications already have significant efficiencies built into them (but may still benefit from Pooled Tables). In some applications, the reporting costs may not be significant enough to see appreciable gains. In others, the candidate reports may not be consecutive and the time to reorganize them may not justify the potential savings. Online applications and ad hoc reports typed at the terminal run single reports on demand and therefore will not invoke pooling.

How Pooled Tables Works

Report processing consists of data retrieval, record processing, and output formatting. When two or more contiguous reports access the same logical database using the same access method, Pooled Tables replaces their individual data retrieval requests with one combined retrieval plan. By reading the database only once, pre-processing the data, and managing the output process, Pooled Tables reduces I/O and CPU costs.

A logical database consists of physical data that can span several files or structures accessed together for a request. The term access method describes the specific technique used to read the data, such as sequential access, indexed access, or keyed search.

The following diagram illustrates processing with and without Pooled Tables:


Standard Reporting
Data
Data Retrieval
Report 1 Read Data
Report 2 Read Data
Report 3 Read Data
Record Processing
Report 1 Local Selection Format Conversions Calculations
Report 2 Local Selection Format Conversions Calculations
Report 3 Local Selection Format Conversions Calculations
Output Formatting
Report 1 Aggregation Formatting
Report 2 Aggregation Formatting
Report 3 Aggregation Formatting
Reporting With Pooled Tables
Data
Data Retrieval
Pooled Tables Read Data
Record Processing
Pooled Tables Global Selection Format Conversions Calculations
Output Formatting
Report 1 Local Selection Aggregation Formatting
Report 2 Local Selection Aggregation Formatting
Report 3 Local Selection Aggregation Formatting

Data Retrieval

Data retrieval is the process of obtaining data from the source. Pooled Tables does not change the physical retrieval process. It builds appropriate calls to legacy databases, generates SQL for relational files, retrieves blocks of sequential data, and retrieves FOCUS database pages. It caches data when possible.

However, instead of issuing separate data requests for each report in the pool, Pooled Tables combines them into a single retrieval plan that reads the data once and contains only those columns needed for the pooled reports. Depending on the data source, this retrieval plan may also apply global selection tests. Database I/O for the pooled reports is the same as for one report. Savings vary directly with the size of the source file and the number of requests that are pooled.

Record Processing

Standard record processing examines one input record at a time, applies selection tests, performs format conversions, and calculates the values of derived columns. The CPU cost to perform these tasks is proportional to the number of input records processed.

In its record processing step, Pooled Tables separates global record selection criteria from local selection criteria. It also performs global format conversions and calculates derived columns for the entire pool. Pooled Tables does not create an extract file or cache data in memory. It processes each record individually and passes the result to each report in the pool.

Selection and Filtering

Pooled Tables globally applies those equality selection tests that are contained in every report in the pool. In addition, it applies all filter screening conditions. This global processing reduces the number of records retrieved from the database and presented to the individual reports.

After completing the global selection process, Pooled Tables presents the remaining input records to the individual reports for local selection. In some cases, the number of input records passed to each report may be greater than the number that would have been presented to each report if it had executed independently. However, the savings in database I/O, along with the efficiencies inherent in global record processing, should outweigh the effects of the additional input. The savings vary with the nature of the requests.

Output Formatting

After record processing, each request handles the records passed to it as if it had retrieved the data itself. It processes local selection tests, aggregation, and output formatting. With Pooled Tables, all reports sort concurrently in memory. CPU use for this phase is directly related to the volume of data selected for the report and is unique to each report.

Pooled Tables does not affect output generation. Report output is generated serially in the order of the requests within the application.

Conditions for Pooling Reports

In order to pool, requests must be contiguous and access the same logical database using the same access method. Requests from separate procedures can pool as long as they execute consecutively.

When a sequence of reports against one database is interrupted by a report from a second database, Pooled Tables executes the reports from the first database before proceeding with the second database. If a later report accesses the first database, Pooled tables does not move it into the first pool because this would change the logical order of processing within the application.

Certain non-reporting commands do not interrupt pooling, even when they are intermixed with reports. For example, commands that create temporary columns do not interrupt the Pooled Tables stream. Certain commands do create temporary boundaries in the pooling process. When it encounters a boundary, Pooled Tables executes all reports in the current pool before continuing to process the application.

Commands that create boundaries include those that require immediate execution, have the potential to update the data, change the source of the data, change the reporting environment, or change the operating environment.

A small class of reports cannot pool because of their syntax. For example, reports that redefine a database field cannot pool because one report may use the redefined value while another needs the database value. A list of conditions that prevent pooling is published in the Pooled Tables new feature documentation.

Tracing Execution

The Pooled Tables trace utility describes how an application is pooled. It displays the size estimates for a report and the decisions that it made based on them. It shows actual input record and output line counts for the reports, specifies why a report did not pool, and identifies any boundary conditions encountered. You can use the information from the trace to fine-tune applications. You may be able to improve performance by simply rearranging the order of reports in an application.

Pooled Tables and Extract Files

A common application tuning technique is to create an extract file that contains a subset of the original data and to execute all subsequent reports from this sequential file. Pooled Tables improves on this technique in several ways.

First, it dynamically determines which reports to pool. You can add reports to your application without preplanning. If you use the extract file technique, you must change the procedure that creates the extract file in order to capture the additional data for the new report.

Pooled Tables retrieves the data for the pool once and presents it via memory to each report. An extract file must be processed multiple times: once to create the file and once for each subsequent report to read it.

For existing applications, you can eliminate the multiple passes against the extract file by pooling the reports that access it. When you create new applications, you no longer need to create an extract file for performance tuning.

Pooled Tables and Relational Tables

In its retrieval plan for accessing relational tables, Pooled Tables issues a simple SQL SELECT request to the RDBMS to retrieve the data for an entire pool of reports. Depending on your environment, data retrieval will either be optimized by the RDBMS or handled by FOCUS. The optimization status affects how reports pool. Performance may actually improve with RDBMS optimization disabled because more reports can be pooled.

Pooled Tables With Optimization Enabled

When data retrieval is optimized by the RDBMS, Pooled Tables cannot anticipate the retrieval plan in advance or assume that the RDBMS will use the same retrieval plan from one execution to the next.

In order to ensure that the RDBMS retrieval engine uses the same optimization logic for each report in the answer set, all reports in a pool must access the same logical database. In addition, all requests against a multi-table relational view must reference the same tables in order to be pooled. For example, if a view contains Table A and Table B, reports that reference columns only from Table A can pool, and reports that reference columns from only Table B can pool. Reports that reference columns from both Table A and Table B can also pool. However, none of the reports in each of these three sets can be pooled with reports from another set.

In order to ensure that the answer set presented to each report in the pool is accurate, reports that invoke SQL aggregation (that is, the generated SQL statements contain the GROUP BY phrase) are not pooled.

Only one answer set is returned for all reports in the pool. Unless all reports select the same data (the optimal case), the answer set for Pooled Tables is larger than for the individual requests. This increase in size may affect performance if it causes the RDBMS to use a table scan for retrieval when it would have used a more efficient technique for each individual report. However, if at least one report in the pool requires a table scan on its own, using a table scan for the pool will not degrade performance.

Pooled Tables With Optimization Disabled

With optimization disabled, reports pool as if they were sequential files. Reports using different tables in a multi-table database pool because FOCUS and Pooled Tables determine the relational structure. Reports that use aggregation pool because FOCUS performs the aggregation, not the RDBMS.

Memory Management

The number of reports that can execute concurrently in one pass of the data source is limited only by the amount of available memory. In planning the memory needs for each report, Pooled Tables uses estimates for the number of input records and output lines. If you do not supply your own estimates, Pooled Tables uses default values set at installation time. While accurate estimates maximize your savings from Pooled Tables, incorrect estimates do not degrade performance unless they are grossly inaccurate. Even so, it is worthwhile to look at report execution histories to determine reasonable estimates.

With Pooled Tables, multiple reports execute concurrently and each requires its own memory area for data storage. To maximize the number of reports that can execute concurrently and to optimize sort performance, Pooled Tables controls the amount of memory allocated to each report for sorting. The amount of memory that Pooled Tables will use is limited by the value of the POOLMEMORY parameter (described in *Ease of Use*) or available memory, whichever is less.

Memory Allocation

Pooled Tables allocates memory simultaneously for all reports in the pool and distributes this memory to each report based on estimates for input records and output lines. This memory would be allocated serially if the reports were executed without Pooled Tables, but its careful pre-planning lowers total memory use over time.

Small reports whose output is less than 256 kilobytes reserve memory equal to the volume of data. Intermediate reports with up to one megabyte of output reserve 256 kilobytes of memory. Large reports reserve sufficient memory to efficiently handle the output. With 16 megabytes of memory available, at least 64 small reports can execute concurrently. Reports with large output require more memory.

The ratio of estimated input records to output lines is called the aggregation ratio. A high aggregation ratio identifies a report that does a lot of aggregation, such as a summary report. For large reports with an aggregation ratio of 50-to-1 or greater, FOCUS sorts and aggregates the data efficiently in memory.

Exceeding Available Memory

Pooled Tables reads the original data only once, even when the memory required for a pool exceeds the amount of available memory.

When it encounters this situation, Pooled Tables subdivides the reports into a series of steps called iterations. During the first iteration Pooled Tables reads the host data source and produces as many reports as possible directly in the available memory. It stages data for the remaining reports in a temporary work file and executes these reports in subsequent iterations using data from the stage file.

The stage file contains data required for reports in the second and subsequent iterations. Data used exclusively in the first iteration is not retained. The temporary file is created only when insufficient memory is available to execute all reports in the first iteration. Data storage is optimized for Pooled Tables and is more efficient than rereading the data from the source file.

Effects of Inaccurate Estimates

Incorrect estimates become acute only when they are grossly inaccurate. If too much memory is allocated, fewer reports execute concurrently. If too little memory is allocated, fewer records are sorted in memory.

While accurate size estimates produce optimal performance, even with inaccurate estimates your reports benefit from reduced database I/O and CPU. Although the report sorting and formatting costs may be higher than for the optimized case, benchmark results show that these losses are small compared to the savings possible.

Critical boundary conditions exist when estimated report output exceeds 256 kilobytes, when report output exceeds 1 megabyte, and when the aggregation ratio exceeds 50-to-1. You can potentially lose performance when estimates fall into one category but actual results occur in another.

Pooled Tables easily forgives poor estimates. The value for estimated lines is ignored if the report does no aggregation; the estimated record count is used in its place. Pooled Tables anticipates values within 10 percent of the actual count and plans accordingly. Estimates outside this margin generate a warning message in the trace utility. When creating a stage file, Pooled Tables counts the number of records for each report and uses this accurate value instead of the estimates.

Ease of Implementation

Pooled Tables is easy to implement. The software is already installed in FOCUS 7.0.8. To enable Pooled Tables and define the Pooled Tables environment, add the following four parameters to the initialization file.

`SET POOLFEATURE = {ON|OFF}` enables Pooled Tables. OFF is the default value.

`SET POOLBATCH = {ON|OFF}` automatically implements Pooled Tables for batch processing. OFF is the default value.

`SET MAXPOOLMEM = n` sets the maximum number of kilobytes of memory above 16 megabytes available for users to set in their sessions. The default is 32,768 (32M). The minimum is 1,024.

`SET POOLRESERVE = n` sets the amount of memory, in kilobytes, to reserve for other modules (such as the SQL/DS Interface). Pooled Tables will be restricted from using this memory. A user can override this value in a FOCUS session. In VM, the default is 1,024. In MVS, the default is 100.

With the exception of POOLRESERVE, users cannot change these settings in a FOCUS session. Changes take effect immediately without compiling, relinking modules, or performing a system IPL.

Ease of Use

Once you have enabled Pooled Tables, you use it by turning it on and supplying estimates for the number of input records and output lines.

Pooled Tables uses these estimates to calculate the memory required for each report. Default estimates are provided by FOCUS. To achieve the greatest benefit, you should provide accurate estimates (within 10 percent). You can choose reasonable values by examining previous execution histories for the reports. Individual users have the option of setting their own estimates either globally or in individual requests.

The following commands add Pooled Tables to your application:

`SET POOL = {ON|OFF}` begins or ends Pooled Tables. OFF is the default value.

`SET ESTRECORDS = n` establishes a global estimate for the number of input records retrieved per report. The default is 100,000. To provide estimates in individual reports, use `ON TABLE SET ESTRECORDS n`.

`SET ESTLINES = n` establishes a global estimate for the number of output lines per report. The default is 0. If no value is given, Pooled Tables assumes that there is no aggregation and that the number of output lines is equal to the number of input records. To provide estimates in individual reports, use `ON TABLE SET ESTLINES n`.

`SET POOLMEMORY = n` sets the maximum memory, in kilobytes, used by Pooled Tables per user session. In MVS, the number represents memory above the 16 megabyte line. In VM, the number represents total virtual memory. The default value is 16,384 (16M). The minimum value is 1,024.

Use global estimates when a group of reports in an application have similar sizes. Use individual estimates when report sizes vary greatly within a pool or to achieve the greatest savings. Accurate estimates provide 20 percent better performance gains in benchmark tests than grossly incorrect values.

The following example pools three reports against the same database. It begins by turning Pooled Tables on and setting global estimates for input records and output lines. The first two reports use these global estimates. The third report uses local estimates. The example concludes by turning Pooled Tables off, at which time retrieval begins for the pool and the output is generated. This application takes advantage of the Pooled Tables default memory allocation set during installation.

```
SET POOL = ON
SET ESTRECORDS = 1000, ESTLINES = 1000
TABLE FILE EMPLOYEE
PRINT LN FN BY DPT
IF HIRE_DATE GE 860101
END
TABLE FILE EMPLOYEE
SUM CURR_SAL BY CURR_JOBCODE
IF CURR_JOBCODE EQ 'A$*'
END
TABLE FILE EMPLOYEE
SUM GROSS BY PAY_DATE
IF PAY_DATE FROM 960101 TO 961231
ON TABLE SET ESTRECORDS 1200 AND ESTLINES 52
END
SET POOL = OFF
```

Benchmarks

Benchmark tests illustrate substantial reductions in I/O, CPU, and elapsed time with Pooled Tables. Test suites represent applications best suited for Pooled Tables: batch reporting against a variety of file types and a variety of report sizes. The number of reports in the test varies to show the advantage of small pools and more significant performance gains of large pools.

Test Conditions

Types of data: Reports were executed using five database types commonly found in FOCUS reporting applications: VSAM, DB2, IMS, FOCUS, and sequential files. The VSAM, FOCUS, and sequential files had identical data; the DB2 and IMS databases contained similar data. The VSAM database was designed for optimal retrieval. The sequential file was blocked for maximum throughput. In order to compare results, data retrieval was sequential for all files except DB2, where retrieval was optimized by the server.

Size of data: Each file contained 500,000 records.

MVS batch environment: The jobs were submitted with a standard job class during periods of low system use to minimize the impact of other jobs running concurrently.

Measurement: Performance statistics for I/O per file, total CPU, and elapsed time were collated from the JES output for each test. Tests were repeated several times to minimize variations in CPU between jobs.

Test plan trial differences: Test suites were executed both with and without Pooled Tables. Each test executed the same type of report, and the number of reports varied from 2 to 25. Each report used different selection criteria. There was no data overlap with the exception reports, since each report was small enough to use independent data. The summary and detail reports selected unique records; there was data overlap only for tests with a large number of reports.

Request Components

Requests printed or summarized several fields, used varied record-selection criteria, and sorted on three fields. They included no formatting or calculations in order to isolate the I/O component of Pooled Tables savings. Requests spanned three size categories:

Size	Description	Pages	Input Records	Output Lines
Small	Exception Report	2	100	100
Medium	Summary Report	20	50,000	1,000
Large	Detail Report	1,000	50,000	50,000

Benchmark Savings Demonstrated With Pooled Tables

The benchmark tests were designed to demonstrate the behavior of Pooled Tables using simple reports and data structures. They are composed of contiguous reports with no restrictions against pooling. The tests reflect an application that is arranged to take the best advantage of pooling opportunities.

Your reporting applications can achieve the same level of results if you keep together as many reports as possible against the same data source. You can usually accomplish this with only minor changes such as merging multiple batch reports or rearranging reports within an application.

Results of the benchmark testing demonstrate the expected performance gains with Pooled Tables:

- Pooled Tables reads the database only once per pool. Database I/O is reduced by a factor of n, where n is the number of reports in the pool.
- CPU savings are proportional to the number of pooled reports. As reports are added to the pool, CPU savings increase by a constant factor: the cost of I/O to read the database once. Savings are consistent based on the number of reports in the pool without regard to the size of the output.

  Pooled Tables yields larger relative CPU gains with small reports, where database retrieval costs are much more significant than formatting costs. Absolute gains remain constant, however, because database I/O costs are not a function of output size. The percentage of CPU improvement for small reports is higher since they save a more significant portion of costs than large reports.

  CPU savings are greater for more elaborate databases. That is, the more complex the type of data source, the better the savings achieved. FOCUS databases show the smallest (but still significant) gains. As complexity increases from sequential and VSAM files to IMS databases, the CPU savings also increase.
- Elapsed time is reduced in the same proportion that CPU is reduced. As reports are added to the pool, elapsed time savings increase by a constant factor. Of course, elapsed time is dependent on many factors outside the control of Pooled Tables.

  Elapsed time savings effectively increase the batch reporting “window” allowing more reports to execute in a fixed amount of time.

Benchmark tests always show an improvement in I/O for pooled requests. They always show CPU improvements and, most of the time, the improvements are substantial.

Applications that do not take advantage of Pooled Tables do not exhibit any CPU degradation or incur additional overhead.

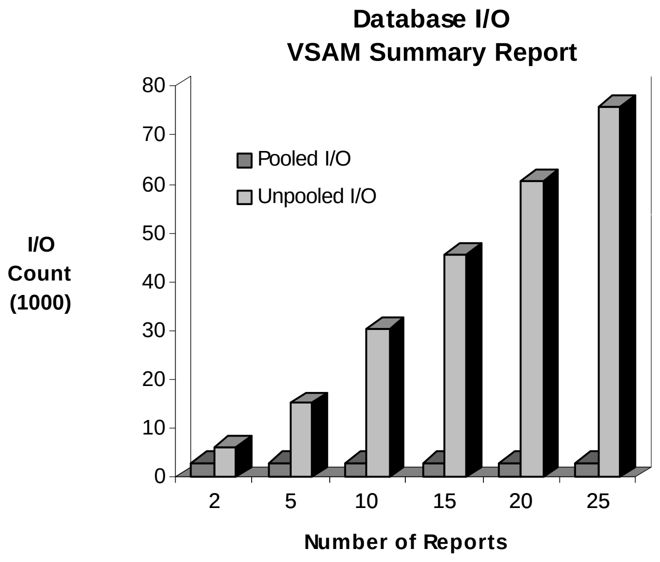
I/O Savings

Pooled Tables reads the database only once per pool.

Database I/O is reduced by a factor of n, where n is the number of reports in the pool.

This test executed the summary report against a VSAM database. The number of reports varied from two to 25.

Pooled Tables read the database only once. With standard reporting, the database was read once for each report. The overhead for reading the file multiple times was eliminated with Pooled Tables. Pooled Tables saved over 3,000 I/Os per report.

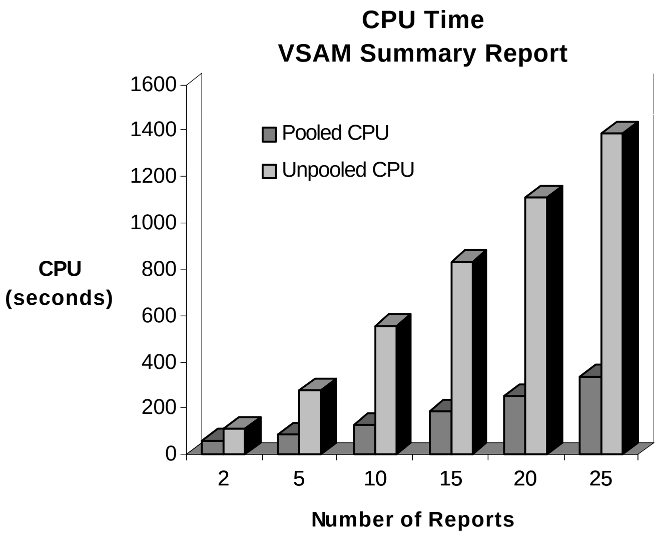


CPU Savings

CPU savings increase with the number of pooled reports.

This test executed the summary report against a VSAM database. The number of reports varied from two to 25.

Pooling two reports saved 45 CPU seconds. As each report was added, the savings grew by a similar amount. With 25 reports, CPU savings approached 18 minutes!



Elapsed Time Savings

Elapsed time is reduced as database I/O and CPU are reduced.

Elapsed time savings increase with the number of pooled reports.

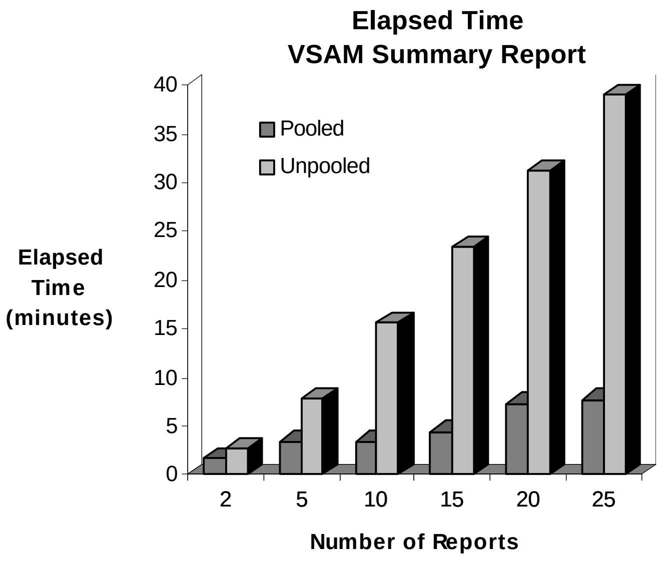
This test executed the summary report against a VSAM database. The number of reports varied from two to 25.

Elapsed time savings are directly related to two factors:

1. It takes less time to physically read the data, minimizing delays due to contention for physical devices.

2. Less CPU is required to process the data once it is read. As CPU decreases, so does elapsed time.

Of course, elapsed time is dependent on many factors outside the control of Pooled Tables.

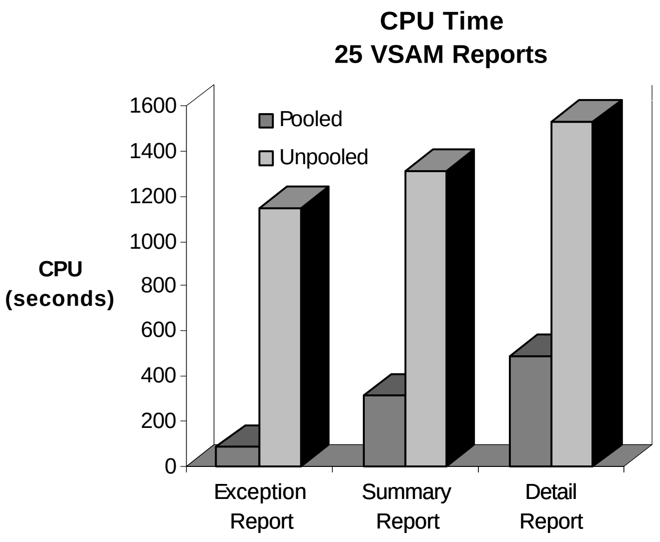


Savings by Report Size

Relative CPU savings are larger for small reports because database retrieval costs are more significant than formatting costs.

This test executed 25 reports of three different sizes against a VSAM database.

CPU savings remain constant regardless of report size when the number of reports in the pool is the same. Total CPU costs rise as report size grows because larger reports sort and format more data. Yet the CPU cost to read the data with sequential access for large and small reports is the same.

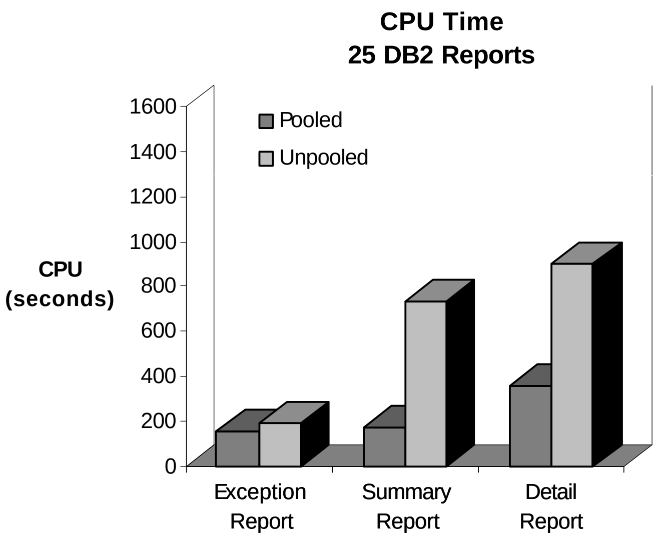


Savings for Relational Tables

CPU is reduced for relational reports, even with RDBMS optimization.

This test executed 25 reports of three different sizes against a DB2 database.

Retrieval costs for optimized requests are quite low. Savings from Pooled Tables are a result of requesting only one answer set, processing the answer set only once, and managing the output process. Pooled Tables savings with non-optimized requests will be even more dramatic when the data is accessed via a table scan.

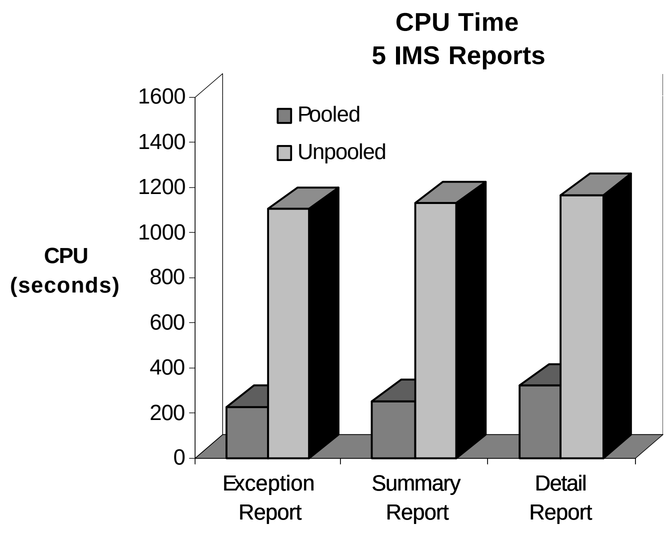


Savings by Degree of Complexity

Relative CPU savings increase with the complexity of the data source.

This test executed five reports of three different sizes against an IMS database.

CPU savings are greater for more complex databases. The harder it is to read the data, the better the savings you can realize with Pooled Tables.



Conclusion

As you can see, Pooled Tables can significantly decrease the costs of your reporting applications and will not increase costs for requests that do not use it. It dynamically analyzes your applications and adjusts to their current requirements.

Using Pooled Tables for your applications reduces database I/O, CPU time related to database I/O, and elapsed time. It works with relational, legacy, FOCUS, VSAM, and sequential data sources.

Pooled Tables is easy to use and configure because it requires no installation, no changes to existing application logic, and no preplanning by a developer or system technician.